Amendments to the Specification

Please replace the paragraph beginning on page 9, line 3, as follows:

Figure 6 is a block diagram of the structure of the ~~XML~~ extensible markup language ("XML") file resulting from Figure 5;

Please replace the paragraph beginning on page 12, line 1, as follows:

Data management system 10, as a component based development framework supports both extensibility and re-use of components. Extensibility means the ability to add new components without having to neither change nor recompile existing system code or the code of existing components. Re-use means the ability to use existing components when building new components or applications. In the data management system, a dataflow application is also a component. Thus, in the data management system, re-use means the ability to use existing components when creating new components (this is also known as composition). The data management system offers a declarative language ~~API~~ application programming interface ("API") to build components as well as a host language class library to customize components and to develop and execute new components.

Please replace the paragraph beginning on page 14, line 1, as follows:

Figures 2A and 2B summarize the lifecycle of the creation and execution of a dataflow application in accordance with a preferred embodiment of the present invention and are useful in understanding the high level architecture hereof. A map component lifecycle starts when the map component developer creates a map component source ("MCS") (MCS file 201), an XML file (Fig. 2A). The MCS file can represent a family of one, in which case the specification is a declarative program indicating a complete dataflow graph, or a family of N, in which case the specification has a declarative part and another part composed of configurable host language procedural components. In any case, the finalized MCS file 201 is a configurable specification

that, given particular bindings of available properties, indicates how internal components are linked to obtain a final, complete, dataflow graph, i.e. a "map."

Please replace the paragraph beginning on page 14, line 13, as follows:

When the developer is done creating the MCS file, then a tool called "map component assembler" 203 is used to generate a map component ("MC") file (MC file 205), also an XML file, which contains a clear text documentation section as well as an encrypted section. MC files 205 are the encrypted image of map components. MCS files 201 cannot be used directly to drive the data management system engine. MC files 205 are used instead. Once an MC file 205 is created, its MCS file 201 is no longer required in all sub-sequent lifecycle processes. Thus, in a way, MCS files are to mc files what java source files are to java class files.

Please replace the paragraph beginning on page 16, line 6, as follows:

If the map represented by the map object 219 has no inputs or outputs then it is a "dataflow application" and can be executed. However, before it can be executed it has to be "prepared" by passing it to the "plan preparer" 221, another tool (Fig. 2B). The plan preparer 221, if everything is ok, then produces a "prepared map object[[.]]" 222. The prepared map object 223 can then be queried with respect to which runtime properties it supports. To proceed with execution the user then gives the prepared map object along with a list of runtime-property-name/runtime-property-value pairs to the executor 14.

Please replace the paragraph beginning on page 38, line 28, as follows:

Figure 17 shows the next level of the design composition (i.e., the internals of text splitter 618). These internals are synthesized using the specifications of the flat file schema parameter 610 and the number of CPUs available at runtime provided by the executing system as CPU count 640. By deferring the synthesis of the text splitter until the runtime environment is known, the optimal dataflow sub-graph for text splitter 618 may be synthesized. In the example depicted in Fig. 17, the available CPU count is four. The character stream input at port 625 passes to

input port 631 of text parser 630 and also to input ports 642 of the four extractors 640. Text parser 630 uses line delimiter 608 and field delimiter 609 parameters to analyze the input character stream and, thus, generates streams of offsets and lengths indicating the position of fields within the strings. This offset and length data appear on composite output port 632 of text parser 630 and are passed individually to ports 641 of the four extractors ~~640~~ 650. By using the offsets and lengths, each processor selects its own data from the character stream, thus dividing the processing effort over the CPUs. Data streams from each extractor ~~640~~ 650 appear on composite output ports 643. The specification of ports 643 can now be returned to the next higher level of the hierarchical map design to enable the synthesis of ports which depended on port 643.